Bitmap Join Indexes vs. Data Partitioning

Ladjel Bellatreche
Poitiers University, France

INTRODUCTION

Scientific databases and data warehouses store large amounts of data ith several tables and attributes. For instance, the Sloan Digital Sky Survey (SDSS) astronomical database contains a large number of tables with hundreds of attributes, which can be queried in various combinations (Papadomanolakis & Ailamaki, 2004). These queries involve many tables using binary operations, such as joins. To speed up these queries, many optimization structures were proposed that can be divided into two main categories: *redundant structures* like materialized views, advanced indexing schemes (bitmap, bitmap join indexes, etc.) (Sanjay, Chaudhuri & Narasayya, 2000) and vertical partitioning (Sanjay, Narasayya & Yang 2004) and *non redundant structures* like horizontal partitioning (Sanjay, Narasayya & Yang 2004; Bellatreche, Boukhalfa & Mohania, 2007) and parallel processing (Datta, Moon, & Thomas, 2000; Stöhr, Märtens & Rahm, 2000). These optimization techniques are used either in a sequential manner ou combined. These combinations are done intra-structures: materialized views and indexes for redundant and partitioning and data parallel processing for no redundant. Materialized views and indexes *compete for the same resource representing storage*, and *incur maintenance overhead in the presence of updates* (Sanjay, Chaudhuri & Narasayya, 2000). None work addresses the problem of selecting combined optimization structures. In this paper, we propose two approaches; one for combining a non redundant structures horizontal partitioning and a redundant structure bitmap indexes in order to reduce the query processing and reduce the maintenance overhead, and another to exploit algorithms for vertical partitioning to generate bitmap join indexes. To facilitate the understanding of our approaches, for review these techniques in details.

Data partitioning is an important aspect of physical database design. In the context of relational data warehouses, it allows tables, indexes and materialised views to be partitioned into disjoint sets of rows and columns that are physically stored and accessed separately (Sanjay, Narasayya & Yang 2004). It has a significant impact on performance of queries and manageability of data warehouses. Two types of data partitioning are available: vertical and horizontal partitionings.

The vertical partitioning of a table T splits it into two or more tables, called, sub-tables or vertical fragment, each of which contains a subset of the columns in T. Since many queries access only a small subset of the columns in a table, vertical partitioning can reduce the amount of data that needs to be scanned to answer the query. Note that the key columns are *duplicated* in each vertical fragment, to allow "reconstruction" of an original row in T. Unlike horizontal partitioning, indexes or materialized views, in most of today's commercial database systems there is no native Database Definition Language (DDL) support for defining vertical partitions of a table (Sanjay, Narasayya & Yang 2004). The horizontal partitioning of an object (a table, a vertical fragment, a materialized view, and an index) is specified using a partitioning method (range, hash, list), which maps a given row in an object to a key partition. All rows of the object with the same partition number are stored in the same partition.

Bitmap index is probably the most important result obtained in the data warehouse physical optimization field (Golfarelli, Rizzi & Saltarelli, 2002). The bitmap index is more suitable for low cardinality attributes since its size strictly depends on the number of distinct values of the column on which it is built. Bitmap join indexes (BJIs) are proposed to speed up join operations (Golfarelli, Rizzi & Saltarelli, 2002). In its simplest form, it can be defined as a bitmap index on a table R based on a single column of another table S, where S commonly joins with R in a specific way.

Many studies have recommended the combination of redundant and non redundant structures to get a better performance for a given workload (Sanjay, Narasayya & Yang 2004; Bellatreche, Schneider, Lorinquer & Mohania, 2004). Most of previous work in physical database design did not consider the interdependence between redundant and no redundant optimization structures. Logically, BJIs and horizontal partitioning are two similar optimization techniques - both speed up query execution, pre-compute join operations and concern selection attributes of dimension tables[1]. Furthermore, BJIs and HP can interact with one another, i.e., the presence of an index can make a partitioned schema more efficient and vice versa (since fragments have the same schema of the global table, they can be indexed using BJIs and BJIs can also be partitioned (Sanjay, Narasayya & Yang 2004)).

BACKGROUND

Note that each BJI can be defined on one or several non key dimension's attributes with a low cardinality (that we call indexed columns) by joining dimension tables owned these attributes and the fact table[2].

Definition: An indexed attribute A_j candidate for defining a BJI is a column A_j of a dimension table D_i with a low cardinality (like gender attribute) such that there is a selection predicate of the form: $D_i.A_j$ θ value, θ is one of six comparison operators {=,<,>,<=,>=}, and value is the predicate constant.

For a large number of indexed attributes candidates, selecting optimal BJIs is an NP-hard problem (Bellatreche, Boukhalfa & Mohania, 2007).

On the other hand, the best way to partition a relational data warehouse is to decompose the fact table based on the fragmentation schemas of dimension tables (Bellatreche & Boukhalfa, 2005). Concretely, (1) partition some/all dimension tables using their simple selection predicates ($D_i.A_j$ θ value), and then (2) partition the facts table using the fragmentation schemas of the fragmented dimension tables (this fragmentation is called derived horizontal fragmentation (Özsu a Valduriez, 1999)). This fragmentation procedure takes into consideration the star join queries requirements. The number of horizontal fragments (denoted by N) of the fact table generated by this partitioning procedure is given by:

$$N = \prod_{i=1}^{g} m_i,$$

where m_i and g are the number of fragments of the dimension table D_i and the number of dimension tables participating in the fragmentation process, respectively. This number may be very large (Bellatreche & Boukhalfa & Abdalla, 2006). Based on this definition, there is a strong similarity between BJIs and horizontal partitioning as show the next section.

Similarity between HP and BJIs

To show the similarity between HP and BJIs, the following scenario is considered[3]. Suppose a data warehouse represented by three dimension tables (TIME, CUSTOMER and PRODUCT) and one fact table (SALES). The population of this schema is given in Figure 1. On the top of this the following query is executed:

```
SELECT Count(*)
FROM CUSTOMER C, PRODUCT P, TIME T,
SALES S
WHEERE C.City='LA'
AND P.Range='Beauty'
AND T.Month='June'
AND P.PID=S.PID
AND C.CID=S.CID
AND T.TID=S.TID
```

This query has three selection predicates defined on dimension table attributes City (City='LA'), Range (Range='Beauty') and Month (Month = 'June') and

Figure 1. Sample of data warehouse population

Customer

RID^C	CID	Name	City
6	616	Gilles	LA
5	515	Yves	Paris
4	414	Patrick	Tokyo
3	313	Didier	Tokyo
2	212	Eric	LA
1	111	Pascal	LA

Product

RID^P	PID	Name	Range
6	106	Sonoflore	Beauty
5	105	Clarins	Beauty
4	104	WebCam	Multimedia
3	103	Barbie	Toys
2	102	Manure	Gardering
1	101	SlimForm	Fitness

Time

RID^T	TID	Month	Year
6	11	Jan	2003
5	22	Feb	2003
4	33	Mar	2003
3	44	Apr	2003
2	55	May	2003
1	66	Jun	2003

Sales

RID^S	CID	PID	TID	Amount
1	616	106	11	25
2	616	106	66	28
3	616	104	33	50
4	545	104	11	10
5	414	105	66	14
6	212	106	55	14
7	111	101	44	20
8	111	101	33	27
9	212	101	11	100
10	313	102	11	200
11	414	102	11	102
12	414	102	55	103
13	515	102	66	100
14	515	103	55	17
15	212	103	44	45
16	111	105	66	44
17	212	104	66	40
18	515	104	22	20
19	616	104	22	20
20	616	104	55	20
21	212	105	11	10
22	212	105	44	10
23	212	105	55	18
24	212	106	11	18
25	313	105	66	19
26	313	105	22	17
27	313	106	11	15